High Level Design

The main component involves a User Interface that is deployed through Amazon Web Services. The UI is a shift scheduler for nurses, which includes a weekly calendar, shift warnings, alternating/updating existing and new shifts, and nurse details. This UI connects to the service layer that handles the business logic. The business logic notifies the user about various potential warnings and details within each shift. The service layer is connected to the API, which is in existence. The API simply calls various functions to the existing database. The database dynamically updates based on the scheduler UI. All of the existing architecture includes Kubernetes/Micro services, stream processor, data science, ETL/data processing. This back-end layer already connects to the API and handles the actual scheduling.

The following diagram was given to us by the Medecipher team as a high-level overview of the project:

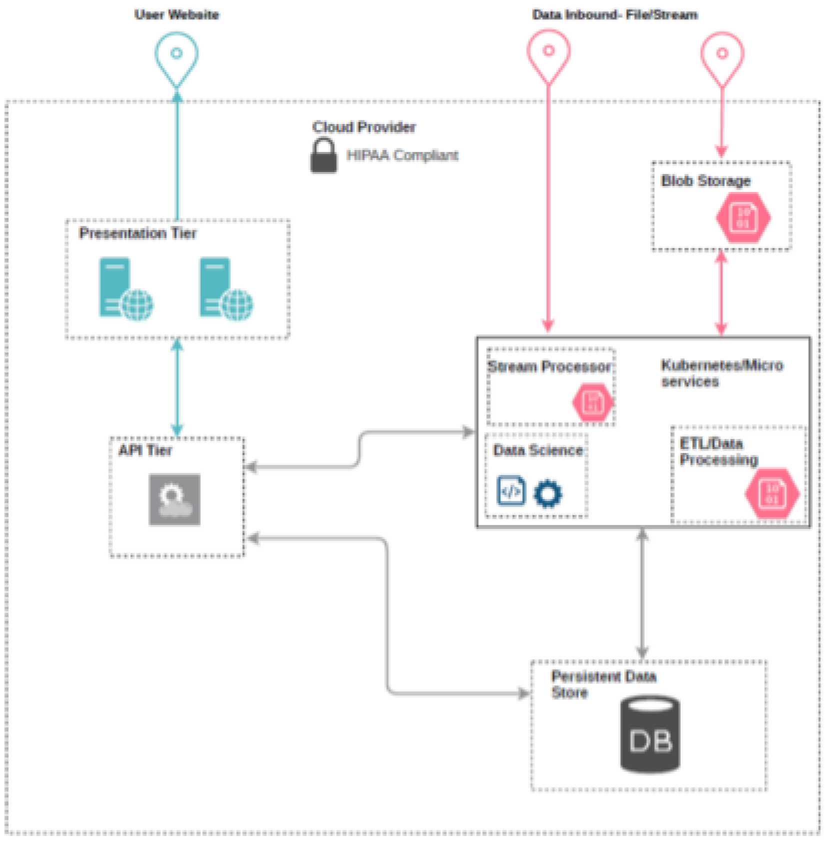

Figure 1: Design Architecture from Medecipher

Detail Design

The architecture of Kubernetes, API, and the database are already developed by Medecipher. The presentation tier is the user interface that is written in AngularJS. The specifics of the presentation layer are displayed in **Figure 2**.

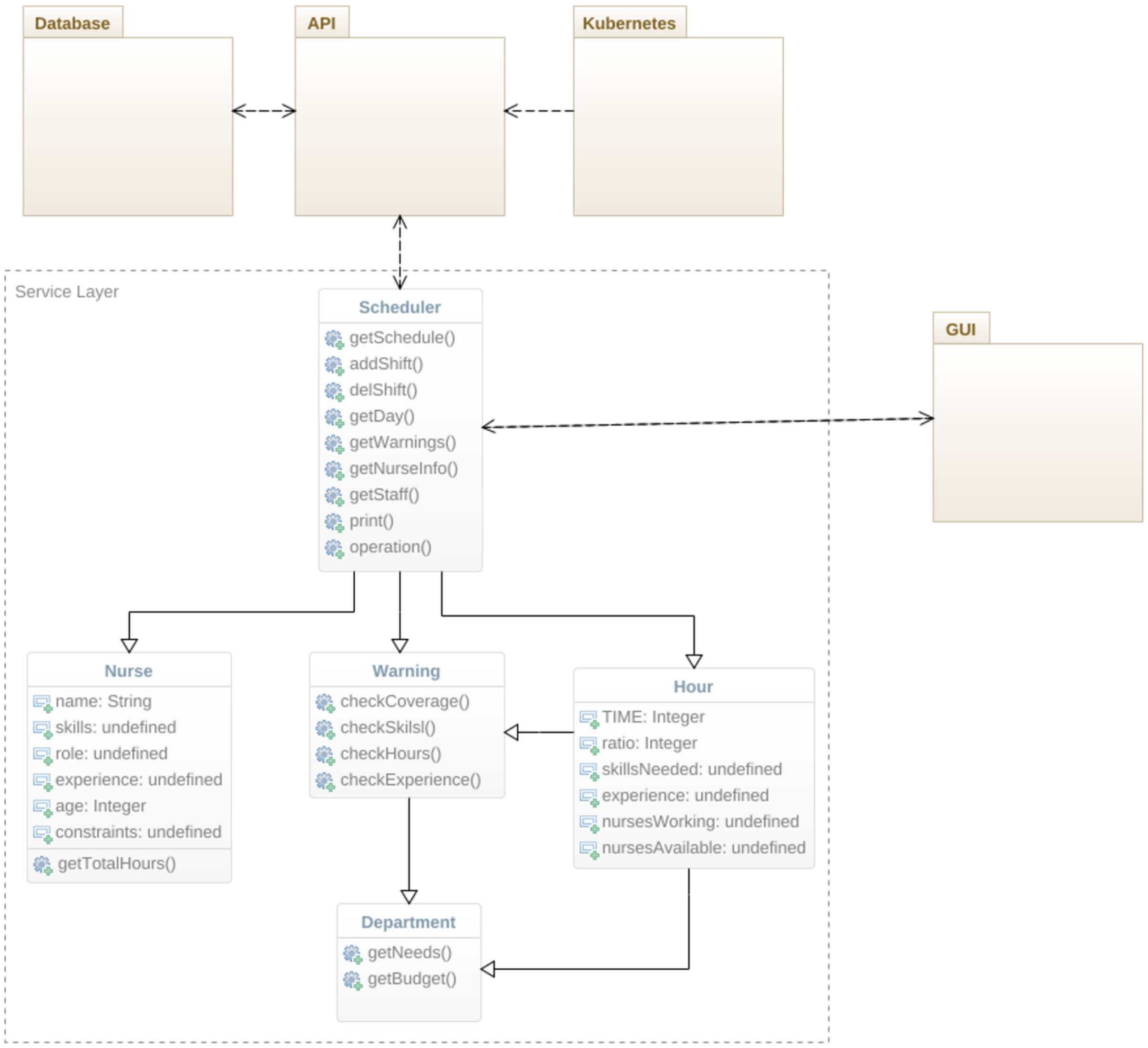

Figure 2: UML Diagram

The UML diagram depicts the details of how the user interface will be created. The various functions within the *Scheduler* class will tell the API what to grab/update from the database.

Example

The user decides to add a shift to the current schedule. The event listener in the *Scheduler* class will make a call to the **API**. The **API** will update the **database** that a nurse has been scheduled for a new shift. The **API** returns the nurses that are available to work for that specific shift from the **database**. The *addshift()* function will handle the logic, determining if the nurse specified from the user meets the requirements for that hour. The *addshift()* function updates *Hour* attributes: nurse to patient ratio, skills, years of experience, nurses available, and nurses working. Based on those calculations, the *addshift()* function will call the functions within the

Warning class to alert the user if the nurse they are scheduling creates any potential issues. No matter what, the nurse will be added and displayed to the schedule since the **API** returns available nurses. The logic in the **service layer** only displays warnings.

Design Architecture Complications/Issues:

- Clarification on API:
    - What are specific calls within our service layer when calling any functions within the *Scheduler* class?
    - What is returned from the API?
        - Does the above example accurately describe what the *data science* aspect is doing (returns available nurses)?
    - Where is the majority of warning logic being handled? (API, Database, new service layer)?
- Clarification on the Database:
    - What type?
    - Does the API connection to the database need any additional logic?
    - Do we have access to the nurse's skills, experience, available hours, total hours worked?
    - Does the data represent a floor or hospital?
- Clarification on UI:
    - Do we want the hospital to input their requirements of nurse ratio, hours worked by each nurse, skill level required, experience years required (or a default)?
    - Does the schedule represent one floor/hospital/unit (because skills/ratio/experiences change based on department)?
    - Is this our main task (since API, database, and data science layer is already created)?
- Clarification on Kubernetes:
    - How do we need to connect/understand this layer?

Design Technical Issues:

- Need to understand *Design Architecture Complications/Issues* to begin technical aspects